Results

guoyang2, bwl3, skavi2, wkolcan2

For our project we implemented Dijkstra's Algorithm and the PageRank algorithm that was made famous by Google. We chose to use the OpenFlights dataset to use for our algorithms and implemented both algorithms using a Breadth First Search traversal.

The first thing that we did was figure out how to parse the data. We started this out by creating a FlightGraph header. In the header we parsed the data by iterating over each line and saving the airport data from each line. We used an airport struct that would have the variables of the name, city, routes, country, latitude, and longitude of the airport. An issue we encountered while building the parser for the dataset was that some airport names had more commas or quotation marks than expected. This led to names of airports being cut off and incorrect information being placed into other variables. In order to correct for this we made sure to check for both excessive quotation marks and commas.

After creating a function that could parse both airport and route data, we began to work on the makefile. We made sure that our makefile would compile the following files: FlightGraph.H, Dijkstra.cpp and Portrank.cpp. We decided early on to try and use some of the newer features from C++17 such as structured bindings and parallel execution policies. In order to compile C++17 code, we switched from llvm 6.0.1 to gcc 9.2.0. We chose this compiler since it was available on EWS machines.

The Makefile and FlightGraph header comprise the backbone of our project, and solidly implementing these early on significantly reduced the work required to implement our two algorithms.

The first algorithm that we worked on was Dijkstra's shortest path algorithm. We decided to work on this one first mostly because we felt that it would be the easier of the two and so we would not have to worry about it later on. The first issue that we noticed is that the OpenFlights dataset does not really have anything inherently that we could use as a weight for each edge. We decided that the most logical weight to assign each edge was distance. We calculated the distance for each route using the latitude and longitude of the origin and destination airports. We decided to go with the calculation of the

great-circle distance in order to get an accurate measurement for distance between coordinates. For this we implemented the Haversine formula that would use our destination and origin coordinates. This method gives accurate distance readings of the earth to within .5% for latitude and .2% for longitude. To start the Dijkstra function we calculated the weights for all the edges. In retrospect, we'd like to change this so it calculates edge weight only when required for the algorithm. This would save time overall if there were vertices that do not have to be visited. From there we implemented Dijkstra's algorithm using a priority queue and visited set. We tag each node we visit with the previous node so we are able to trace back our path. We then save the path in a vector and print out the directed path to the terminal. Overall we are satisfied with our Dijkstra's implementation because it works relatively fast in comparison to the size of our dataset. One interesting discovery was that some smaller airports only connected to other smaller airports so there were no viable paths for certain pairs of airports.

For pagerank, we decided to use the iterative algorithm. To do this, we made an adjacency matrix from our FlightGraph object. Unfortunately this is quite large (~500MB), but our runtime was not too badly impacted. This adjacency matrix is then manipulated in such a way that R, the output pagerank vector, is the principal eigenvector. We initialize R to a random unit vector. The manipulated adjacency matrix is then iteratively multiplied into R, and eventually converges to the correct R. The most difficult bit in the implementation of this algorithm was understanding the pagerank algorithm. Implementation was actually fairly straightforward with the STL. One discovery that we made while implementing pagerank is that it doesn't necessarily rank popular airports the highest. Since the data only gives us the # of routes and not the frequency. Therefore more routes gives you more points even if you are a smaller airport running a lot of flights but it's just the same route.

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Haversine Formula

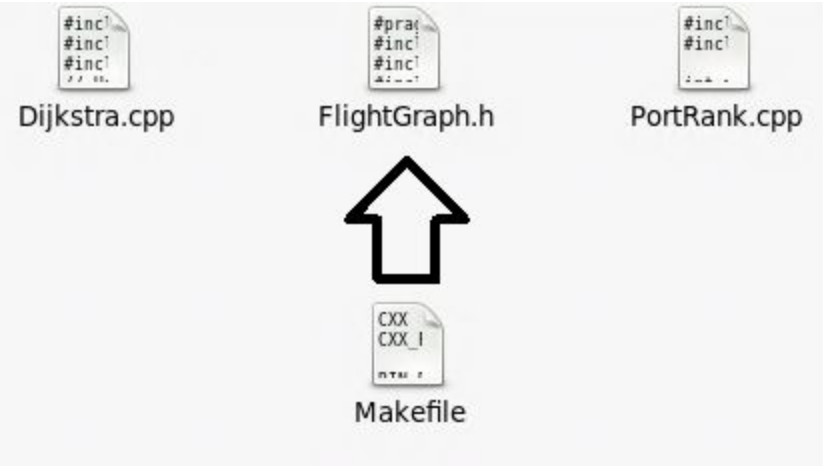
```
[wkolcan2@linux-38 finalproject]$ ./bin/Dijkstra
Using data/airports.dat for airports file and data/routes.dat for routes file.
Finding shortest path from Chicago O'Hare International Airport to Emerald Airport.
Chicago O'Hare International Airport -> Los Angeles International Airport -> Brisbane International Airport -> Emerald Airport
```

Dijkstra's Output

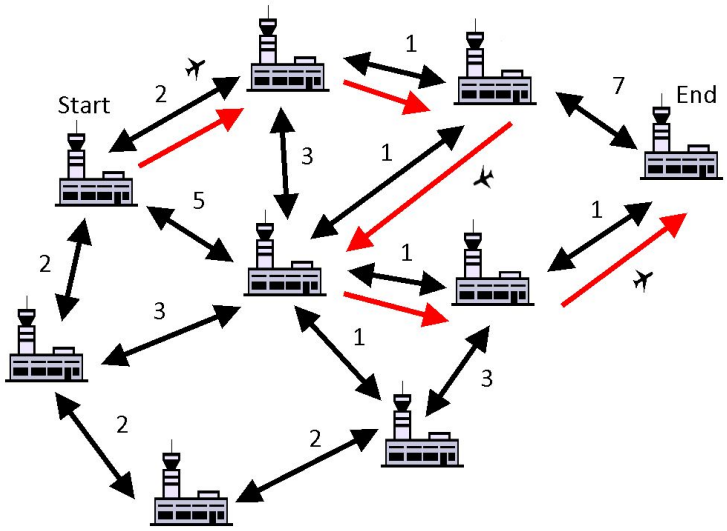
```
Using data/airports.dat for airports file and data/routes.dat for routes file.

Frankfurt am Main Airport: 0.235467
Charles de Gaulle International Airport: 0.231052
Amsterdam Airport Schiphol: 0.227918
Atatürk International Airport: 0.227044
Hartsfield Jackson Atlanta International Airport: 0.220474
Beijing Capital International Airport: 0.206679
Chicago O'Hare International Airport: 0.206598
```

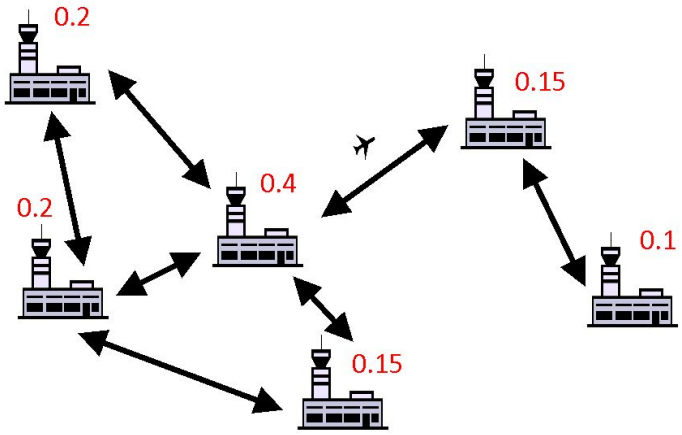
PageRank Output



Makefile Visualization



Dijkstra Visualization



PageRank Visualization